

Maekawa's Algorithm

- Difference with respect to previous algorithms:
 - a site does not request permission from every other site but only from a subset - called *request set*
- The requests sets of any two sites have at least one site in common:

$$\forall i \forall j : 1 \leq i, j \leq N :: R_i \cap R_j \neq \emptyset$$

- The basic idea is that each pair of sites is going to have a third site mediating conflicts between the pair

Maekawa's algorithm steps

- Requesting the CS
 - S_i sends a message **REQUEST**(i) to all the sites in R_i
 - when S_j receives a request from S_i , it returns a **REPLY** to S_i if it has not sent a **REPLY** since receiving the latest **RELEASE** message. Otherwise the request is enqueued.
- Executing the CS
 - A site S_i executes the CS only after receiving **REPLY** messages from all the sites in R_i
- Releasing the CS
 - When done, a site S_i sends a **RELEASE** message to all the sites in R_i
 - When a site receives a **RELEASE** message, it sends a **REPLY** message to the next site waiting in the queue and removes it

Construction of the request set

- The requests sets are constructed to satisfy the following conditions:
 - $\forall i \forall j : 1 \leq i, j \leq N :: R_i \cap R_j \neq \emptyset$
 - necessary for correctness
 - $\forall i : 1 \leq i \leq N :: S_i \in R_i$
 - necessary for correctness (note: this condition, like the need for FIFO comm., is really needed only in the extended version of the algorithm)
 - $\forall i : 1 \leq i \leq N :: |R_i| = K$
 - all R_i have equal size, so all sites do equal work to access the CS
 - Any site S_i is contained in K of the R_i s
 - the same number of site is requesting permission from each site

More on the request set

- All the previous conditions are satisfied if N can be expressed as:

$$N = K(K - 1) + 1$$

(examples: $N = 3$ and $K = 2$, $N = 7$ and $K = 3$, etc.)

– Note that, for large N , $K \approx \sqrt{N}$

- Otherwise one of the last two conditions must be relaxed
 - for example, $|R_i| = K$ not longer true for all i

Notes on Maekawa's algorithm

- Performance:
 - $3\sqrt{N}$ messages are needed for execution of the CS
 - synchronization delay is $2T$
- Problem: the algorithm is deadlock prone!
 - there is a variant of the algorithm that can prevent the deadlock by using a priority-based preempting scheme
 - this variant requires additional messages (up to $5\sqrt{N}$)

Revisiting Vector clocks

- The timestamp C_i of an event a is a vector of length n
 - $C_i[i]$ is P_i 's own logical clock
 - $C_i[j]$ is P_i 's best guess of logical time at P_j 's
- Implementation rules:
 - events a and b are on same process: $C_i[i] = C_i[i] + d$
 - a is the sending and b the receiving of a message m :
 $\forall k, C_j[k] = \max(C_j[k], t_m[k])$

Vector clock: timestamp comparison

- Vector timestamps can be compared in the obvious way:
 - $t^a = t^b$ iff $\forall i, t^a[i] = t^b[i]$
 - $t^a \neq t^b$ iff $\exists i, t^a[i] \neq t^b[i]$
 - $t^a \leq t^b$ iff $\forall i, t^a[i] \leq t^b[i]$
 - $t^a < t^b$ iff $(t^a \leq t^b \wedge t^a \neq t^b)$
- Important observation:
 - $\forall i, \forall j : C_i[i] \geq C_j[i]$

Causally related events

- In a system with vector clocks:
 - $a \rightarrow b$ iff $t^a < t^b$
- Practical consequence: by comparing vector timestamps we can tell if two events are causally related:
 - $t^a < t^b \Rightarrow a \rightarrow b$

Proof Outline

- Proof Obligation

(1) $a \rightarrow b$ implies $t^a < t^b$

(2) $t^a < t^b$ implies $a \rightarrow b$

Cases to Consider

(a) a and b on the same process

(b) a and b on different processes